FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Riina Annunen
Anssi Valjakka

SPACE GREETER: A LOW-POWERED FACIAL RECOGNITION DEVICE FOR PUBLIC SPACES

Bachelor's Thesis
Degree Programme in Computer Science and Engineering
August 2021

Annunen R., Valjakka A. (2021) Space Greeter: a Low-Powered Facial Recognition Device for Public Spaces. University of Oulu, Degree Programme in Computer Science and Engineering, 41 p.

ABSTRACT

Facial recognition is becoming more common all the time with its potentials and with its threats. It's getting more accurate and, it has gotten multiple use cases by far, for example in the field of security. As the technology has matured and highly efficient low-cost computers have been developed, it is now possible to implement an affordable real-time facial recognition system. This thesis contains our implementation of a facial recognition software that aims to recognise person's face and attach it to a name from a database in order to greet them by name. In addition, this thesis contains the evaluation of the implementation and possible future work.

We examine existing programs and libraries in order to determine which would work in our use case. For facial recognition, OpenCV library was deemed the best option for its straightforward implementation and a broad range of functions, basic HTML and PHP based website for user interface and MySQL for storage. Testing system recognition speed and accuracy in real-life deployment was not possible due to covid-19, but a minor test with fewer participants was concluded. Further testing for face recognition performance was done with a FERET-database.

Our tests show that real-time facial recognition with a Raspberry pi 4B as a platform is possible with decent accuracy and speed. In good lighting conditions and with a high confidence level, our algorithm achieved a 2-second delay for face recognition. In theoretical testing using a FERET dataset of 994 persons, our algorithm achieved an accuracy of 93.84% and a delay of 360-milliseconds when easier images were used. With more difficult and realistic images, the accuracy was 52.80% and delay 370-milliseconds. Nevertheless, our implementation could be enhanced further, and we discuss the possible means in the summary chapter.

Keywords: Face recognition, OpenCV, Raspberry pi, Local Binary Patterns, FERET.

Annunen R., Valjakka A. (2021) Space Greeter. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 41 s.

TIIVISTELMÄ

Kasvojentunnistus yleistyy jatkuvasti samalla luoden uusia mahdollisuuksia, mutta myös erilaisia uhkia. Kasvojentunnistusohjelmien tarkkuus paranee jatkuvasti ja ympärillämme on paljon eri käyttökohteita näille tarkoille ohjelmille, esimerkiksi turvallisuusalalla. Kasvojentunnistuksen parantuneen tarkkuuden sekä edullisten tietokoneiden kehityksen myötä myös edulliset reaaliaikaiset kasvojentunnistusohjelmat ovat mahdollistuneet. Opinnäytetyössämme esittelemme oman implementaatiomme kasvojentunnistusjärjestelmästä, joka osaa tervehtiä henkilöitä heidän nimillään.

Tutkimme työtämme varten olemassa olevia ohjelmia ja kirjastoja selvittääksemme mitkä niistä soveltuisivat parhaiten käyttötarkoitukseemme. Päätimme käyttää kasvojentunnistukseen OpenCV-kirjastoa sen helppokäyttöisyyden ja runsaiden ominaisuuksien vuoksi. Käyttöliittymän toteuttamiseen käytimme HTML ja PHP -kieliä sekä tietokantana toimi MySQL. Järjestelmän laajamuotoista testaamista julkisella paikalla emme voineet toteuttaa Covid-19 -viruksen vuoksi, mutta suppeampi testaus suoritettiin muutamalla koehenkilöllä. Lisäksi kasvojentunnistuksen nopeutta ja tarkkuutta mitattiin käyttäen FERET-tietokantaa.

Testimme osoittavat, että kasvojentunnistus on mahdollista toteuttaa kohtuullisella tarkkuudella ja nopeudella Raspberry Pi 4B -tietokonetta käyttäen. Hyvissä valaistusolosuhteissa algoritmimme saavutti kahden sekunnin viiveen kasvojentunnistamiseen. Teoreettisilla testeillä käytettäessä FERET-tietojoukkoa 994:stä ihmisestä algoritmimme saavutti 93.84%:n tarkkuuden 360-millisekunnin viiveellä helpommilla kuvilla. Hankalampia kuvia käytettäessä tarkkuus oli 52.80% ja viive 370-millisekuntia. Yhteenvedossa tarkastelemme kuinka implementaatiotamme voisi kehittää edelleen.

Avainsanat: Kasvojentunnistus, Raspberry pi, paikalliset binäärimallit, FERET.

TABLE OF CONTENTS

FOREWORD

The purpose of this thesis was to design and implement a Raspberry pi based system for recognizing and greeting people by name. We would like to thank Denzil Ferreira for helping us with both technical and writing side of our work, as well as providing us with necessary equipment.

Portions of the research in this paper use the FERET database of facial images collected under the FERET program, sponsored by the DOD Counterdrug Technology Development Program Office.

Oulu, August 8th, 2021

Riina Annunen
Anssi Valjakka

LIST OF ABBREVIATIONS AND SYMBOLS

CNN	convolutional neural networks
CPU	Central Processing Unit
CSS	Cascading Style Sheet
DOM	Document Object Model
FERET	The Face Recognition Technology
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
SQL	Structured Query Language
LBP	Local Binary Pattern
LFW	Labeled Faces in the Wild
AFLW	Annotated Facial Landmarks in the Wild
AFLW2000-3D	68-point 3D facial landmark annotated Facial Landmarks in the Wild
300W	Automatic Facial Landmark Detection in-the-Wild Challenge
PHP	Hypertext Preprocessor
FPS	Frames Per Second
URL	Uniform Resource Locator

1. INTRODUCTION

Before technology's evolvement, we have been identifying people with fingerprints, DNA and dental anatomy. With today's technology, it is possible to do accurate identification with cameras and using facial recognition software along with it. There are many situations where facial recognition is the most convenient way to do person identification. For example, it is a good way to count how many people there are at an event or to detect who is in a building if the count by hand would take too long. For example, in Omnia's vocational school in Espoo, they tried monitoring pupils attendance rates in classes with facial recognition technology [1].

This technology can also be used in security systems, such as unlocking something if the face is connected to a person who has the authority to access it. Unlocking your smartphone with your face is one example of this kind of security technique. Another good example application for facial recognition is identifying patients in hospitals to prevent medical errors such as FA6 med system, which has 99% accuracy and it takes only 1 second for the system to identify a person. With the FA6 med system, doctors are able to access patient's medical information even if the patients are unconscious [2]. Smart homes can also benefit from this technology for example by using Google Nest Hello which is a doorbell that tells you who is at the door [3]. Figure 1 shows this doorbell. More use cases for facial recognition are discussed in 2.

In our work, we implement a system that can recognize a person by taking a picture of one's face. As a computer, we use a low powered and affordable Raspberry Pi 4B and its camera module to take the pictures for the facial recognition software which is running on the Raspberry Pi.

First, we teach our software to detect faces by giving it multiple images of faces. Then it compares the pictures it takes to the database of faces to see if there is a face in the pictures it just took. Suppose we want to identify the person in the picture, we have to get some geometrical data from the picture, for example, distance between eyes or distance between eyes and nose. After this, we compare it to the data stored in a database and see if we have a match.

There are many different methods for facial recognition. One of the oldest methods of facial recognition is to use extracted geometrical features from a face, which is a fast and robust method. Another technique for facial recognition is the eigenface method which extracts relevant information from the face and then encodes it. The eigenface method learns new faces well with good accuracy, but it has some defects such as sensitiveness to changes in illumination [4]. Eigenfaces may discard some important data because it does not take classes into account. Fisherfaces is a method that clusters the same classes, and it is a good technique to extract the most important data while ignoring non differentiating changes such as lighting [5]. Even though fisherfaces and eigenfaces are good methods, they do not perform as well as local binary patterns when only a few images exist. Local binary patterns does better on FERET tests, and it is more usable in real-time applications [6].

In addition to these three methods, there are also methods that utilize 3D modelling. With these 3D methods, we take landmarks from images and calculate their 3D coordinates. These methods are good if the face is not always facing the camera [7]. These methods are described more thoroughly in 2.

In this work, we introduce different methods of facial recognition, which are in chapter 2. We also implement our own facial recognition system with Raspberry Pi 4 and its camera module. The implementation is described more thoroughly in chapter 3. In chapter 4 we discuss the evaluation of our implementation, and in chapter 5 we give a summary of our work and discuss possible future work that could be done to add new features or improve the existing ones.

Figure 1. Google Nest Hello [8]

1.1. Contribution

The first part of our work consists of researching different use cases and facial recognition methods. Anssi Valjakka researched the different methods and Riina Annunen the different use cases. The implementation part was divided in a way that Anssi Valjakka focused on the facial recognition model, as well as its optimization. Riina Annunen was in charge of the hardware side, the user interface and the database. Even though the implementation was divided, the designing was mainly done together. The theoretical testing was performed by Anssi Valjakka and the hands-on evaluation of the system as a whole by Riina Annunen.

2. RELATED RESEARCH

2.1. Different Technologies for Facial Recognition

2.1.1. Face Recognition Technology Program

The Face Recognition Technology (FERET) program was founded to provide both datasets and standards for testing facial recognition systems, making it possible to compare different recognition algorithms and their strengths and weaknesses. The dataset contains images of subjects with varying expression, lighting, scale and pose, making it more similar to real-life use cases [9].

2.1.2. Geometrical Features

One of the oldest methods for facial recognition is using geometrical facial features like the distance between eyes, location of the tip of the nose, et cetera. This method is fast and robust since features can be extracted even from a low-resolution image, and the resulting feature vector takes very little space. However, recognition accuracy is not sufficient for larger datasets. Facial feature-based recognition can be used as a screening step for more accurate and heavier recognition methods [10].

2.1.3. Eigenfaces

Another technique is the eigenface approach, which aims to extract relevant information from a face image, encode it efficiently and compare it to an encoded image with other similarly encoded images. This method makes learning new faces easy and provides accurate recognition with some caveats. The background should be removed to improve recognition, accurate estimation of head size is crucial, recognition in angle is complex, and eigenfaces is sensitive to changes in illumination [4].

2.1.4. Fisherfaces

Even though eigenfaces is an excellent method for representing data to maximise variance, it does not consider classes. Not considering classes may lead to discarding important and emphasising irrelevant data, for example, change in illumination. Fisherfaces applying linear discriminant analysis uses classes to represent the data in a way that clusters the same classes as tightly together as possible and maximises the distance to other classes. This method is great for extracting only meaningful data from images and ignoring things like changes in lighting [5].

2.1.5. Local Binary Patterns

While eigenfaces and fisherfaces fare well when the database includes multiple images of subjects, they have problems if only a few images exist. Local binary patterns can be used to extract local features or micro-patterns from images. The method outperforms other recognition approaches on FERET tests, including testing the robustness of the method against different facial expressions, lighting and ageing of the subjects. LBP-based recognition is also straightforward and fast, improving its usability in real-time applications [6].

2.1.6. Compensating Changes in Illumination

Multiple methods have been proposed to combat illumination conditions, including edge maps, image intensity derivatives, and convolving images with 2D Gabor-like filters. None of these representations is sufficient by themselves to overcome image variations because of a change in the direction of illumination [11].

2.1.7. Handling Head Rotation with 3D Model

Even though illumination changes have been more or less dealt with eigenfaces and LBP:s, head rotation still poses a challenge for facial recognition algorithms. One way to combat this is by creating a 3D reconstruction of the face to counter differences in poses. This method uses 68 facial landmarks detected from an image and calculates their 3D coordinates using a reference face 3D model. Depth optimisation and joint dept-appearance optimisation are used next to improve the quality of the model. The resulting method is easily extendable, requires no manual interaction, is robust to illumination, expression changes, occlusions and viewpoint differences, and is efficient, taking only seconds to estimate depth. These 3D models can then be positioned in a way that is optimal to recognition the algorithm at hand [7].

2.1.8. Multi Step DeepFace Process

To improve existing facial recognition solutions on large datasets, a six-step process DeepFace was proposed: detecting, 2D aligning, 3D aligning, frontalizating, representing and classifying an image. For detection, six fiducial points, the centre of the eyes, the tip of the nose and mouth locations, are used to approximately scale, rotate and translate the face image into six anchor locations to improve recognition accuracy. Following additional 67 fiducial points are located and mapped on a generic 3D face shape model to warp a frontalized view. This frontal image is fed to a deep neural network for representation which is then normalised to reduce sensitivity to illumination changes. Classifying accuracy on Labeled Faces in the Wild dataset reached 97.15%, nearing human performance of 97.53%. Without frontalization, accuracy was 94.3%, without any alignment 87.9% was reached, showing how much image preparation affects final performance [12].

2.1.9. Frontalizing Face Image to Improve Accuracy

Previous work to frontalize an image have been made using a rough approximation of the 3D surface of the face and generating a new view from this surface. Even though intuitive, this approach is shown to be detrimental for accurate feature comparison. Another approach is to produce frontalized faces employing a single unchanged 3D shape with all query photos. Despite a face shape that can be very different from the true shapes, the resulting frontalization lose little of the identifiable features. Being highly aligned allows for easier comparison of appearances across faces improving accuracy over previous solutions [13].

2.1.10. Deep Convolutional Network for Face Detection

In another attempt to improve face detection and alignment accuracy, and performance, a deep coarse-to-fine convolutional network was carefully designed to exploit the inherent correlation between detection and alignment. The framework comprises three stages: First is a proposal network that obtains multiple facial window candidates, next is a refined network that rejects false candidates and performs calibration, and the last one is similar to the second one but aims to identify face regions and outputs five facial landmark positions. The resulting method achieves superior accuracy over the state-of-the-art methods on the challenging face detection datasets and benchmark and WIDER FACE benchmarks for face detection while keeping real-time performance [14].

2.1.11. Fitting a 3D Model to Recognise When Yaw Is Greater than 40 Degrees

While the alignment issue with moderate poses has been solved, poses with yaw greater than 40 degrees cause issues because of self-occlusion, as fiducial points cannot be located. To solve this, a 3D Dense Face Alignment is proposed. This method fits a 3D morphable face model to image with cascaded CNN by minimising the differences between the image and model appearances, to solve the self-occlusion problem. A face profiling algorithm is also proposed to synthesise face appearances in profile view, providing abundant samples for training. Experiments show its state-of-the-art performance in AFLW, AFLW2000-3D and 300W datasets. [15].

2.1.12. Single Deep Convolutional Network FaceNet

Even though most approaches to facial recognition have used intermediate bottleneck layers for image processing, a single deep convolutional network FaceNet can be trained to optimise the embedding itself directly. This method combined with training using triplet loss to minimise the distance between the same identities and maximise to different identities improves recognition efficiency and accuracy. Record-breaking classification accuracy of 99.63% was reached on Labeled Faces in the Wild (LFW). Caveats of this approach are the need for an extensive training dataset and high CPU requirements [16].

2.1.13. Open Source Recognition Tool for Developers and Researchers

To make it easier to implement a computer vision application, FaceNet based OpenFace was developed. It is an open-source tool intended for computer vision and machine learning, providing tools for a landmark, head pose, action unit and gaze detection in real-time, and a messaging system allowing easy integration. This makes it easy to build real-time interactive applications that rely on various facial analysis subsystems [17].

2.1.14. Video Based Facial Recognition

Most facial recognition solutions focus on images, but video-based face recognition provides challenges to facial recognition, as video quality is usually worse than that of still images and face images are often small. On the other hand, a video provides more data, making it possible to construct 3D models for the faces as well as compare recognition of multiple pictures to improve accuracy [18].

2.2. Use Cases

As there are many facial recognition methods, there are many use cases for as well. Five different use cases are discussed in this chapter.

2.2.1. Door Unlocking

For example, in reference to article 16, Thulluri Krishna Vamsi, Kanchana Charan Sai and Vijayalakshmi M built a system where they could unlock a door with a Raspberry Pi and an electromagnetic lock using facial recognition. A web camera was attached to the Raspberry Pi, and it took pictures using the Open-CV platform of the person standing in front of the camera. Then it would send the picture to a database and compare it to the admin's picture in the database. If it was a match, the door would open [19].

2.2.2. Patient Identification

In reference to article 17, a group created a mobile app to create a safe patient identification program. They used Java as the programming language and as a facial recognition engine Oezsoft Inc. The main idea is that doctors can access patient's medical records, examinations and prescriptions, appointments and ethics statements by scanning patient's faces with this application using the mobile phone's camera. The facial data is stored as 3-dimensional vectors and, it is connected to the patient's other information. This application was tested by 62 patients, and the accuracy of the facial recognition was 99% even though some of the patients have had surgery[20].

2.2.3. Help for the Visually Impaired

Laurindo Britto Neto, Felipe Grijalva and associates have also built a facial recognition system for medical purposes. They created a system that would help visually impaired users. They used Microsoft Kinect sensor with Microsoft Face Tracking SDK for Kinect. The device is attached to a helmet, and it consists of three modules: face detection module, 3D audio module and face recognition module. Kinect is detecting faces and then estimates the location of the person. When a face is detected, it tries to recognise the face. After this, it will create a sound at the location where the person is, and if the person is recognised, it will tell the person's name. The best accuracy rate they got during tests was 94.26%[21].

2.2.4. Identification in Exams

Another excellent use case could be, for example, checking identities in exams. In reference to article 19, Arief Agus Sukmandhani and Indrajani Sutedja created a prototype for this purpose. In their work, they used Eigenface to train the system with pictures taken with a webcam. The application itself works on a desktop and is built with Python. They also used Emgu CV-Library and SQLlite database. Sukmandhani and Sutedja ran the tests with this prototype, and as a result, the face recognition was not very accurate. For example, glasses led to not recognising the student[22].

2.2.5. Student Attendance Rate Monitoring

In addition to the so far discussed purposes, we can, for example, keep track of student attendance rates, recognise facial expressions or identify the gender of a person. Guangjun Liao, Wei Chen and Yaoxin Wu tested the latest. In this test, they used a public facial database and a small collected 3D database of faces. As a device, they used Kinect and random forest algorithm as a gender recognition algorithm. They ran an experiment with the Kinect, collecting 3D information of faces. The average correct rate was 83.59%, and the error rate was 16.41%[23].

Table 1. Face recognition methods

Method	Reference
Geometrical facial features	[10]
Eigenface	[4]
Fisherface	[5]
Local binary patterns	[6]
DeepFace	[12]
FaceNet	[16]

2.3. Possible and Existing Threats with Facial Recognition

Even though facial recognition can be used for the good of people, it can still be used for malicious purposes as well. It does not even have to be someone's bad intentions, but if the software makes a mistake or has been tricked, the outcome can be severe and even dangerous.

Face detection technology has many vulnerabilities. The technology can be tricked with masks, pictures, or video material [24]. If some of these spoofs are successful, a person can, for example, gain access somewhere he or she would not normally have access.

In Erdogmus and Marcel's research, they tried to spoof a 2D face recognition with 3D masks, as shown in 2. The test was successful because 66,07% of the spoofing attacks succeeded. This was when they were not using LBP-based countermeasures.

Even though the recognition can be spoofed, there are plenty of countermeasures to prevent this. These are divided into three groups: motion, texture and detection analysis [25].

Figure 2. 3D masks used in Erdogmus and Marce's research [25]

2.4. Libraries

2.4.1. JQuery

JQuery is an open-source lightweight Javascript library. JQuery has tools for creating animations, event handling, HTML Document Object Model (DOM) manipulation. HTML DOM is the standard model for all the HTML documents, and this model defines every HTML element as an object. [26, 27]

JQuery also enables the manipulation of Cascading Style Sheets (CSS). CSS document defines the outlook of an HTML page.[28] All these tools make Javascript

programming easier and more simple. In addition to these tools, jQuery has multiple different plugins for different kinds of tasks [29].

2.4.2. OpenCV Library

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV was built to provide a common infrastructure for computer vision applications. The library has more than 2500 optimised algorithms, including a comprehensive set of classic and state-of-the-art computer vision and machine learning algorithms. In this application, these algorithms are used to detect and recognise faces.
Particularly LBPHFaceRecognizer (Local Binary Pattern Histogram Face Recognizer) was used to implement the core functionality of this application. [30]

2.5. Software

2.5.1. MySQL

MySQL is an open-source database management system distributed by the Oracle Corporation. SQL at the end of the name stands for "Structured Query Language". This is the standard language used for database manipulation, and it has been defined by the ANSI and ISO standards. [31, 32]

2.5.2. Apache

Apache HTTP (Hypertext Transfer Protocol) server is an open-source project developed and maintained by a group of volunteers. This project is a part of the Apache Software Foundation. Apache allows people to build their own web pages, experiment and learn for free [33].

3. IMPLEMENTATION

This paper's implementation can be roughly separated into two different parts. The first part is the hardware side, and the second one is the software side. Both parts are described more thoroughly in the following two sections of this chapter.

3.1. Hardware

The hardware side consists of two primary devices: Raspberry Pi 4B and Raspberry's camera module. The Raspberry Pi runs all the needed scripts and software, and the images are taken with the camera module discussed. See chapter 3.9 for a more accurate system setup description. Also, a keyboard is attached to the Pi to make human-computer interaction possible.

Later on, a cooling case with a dual-fan was installed after the first Raspberry Pi stopped working since the long-running of the software overheated the Raspberry Pi. The cooling case can be seen in figure 3.

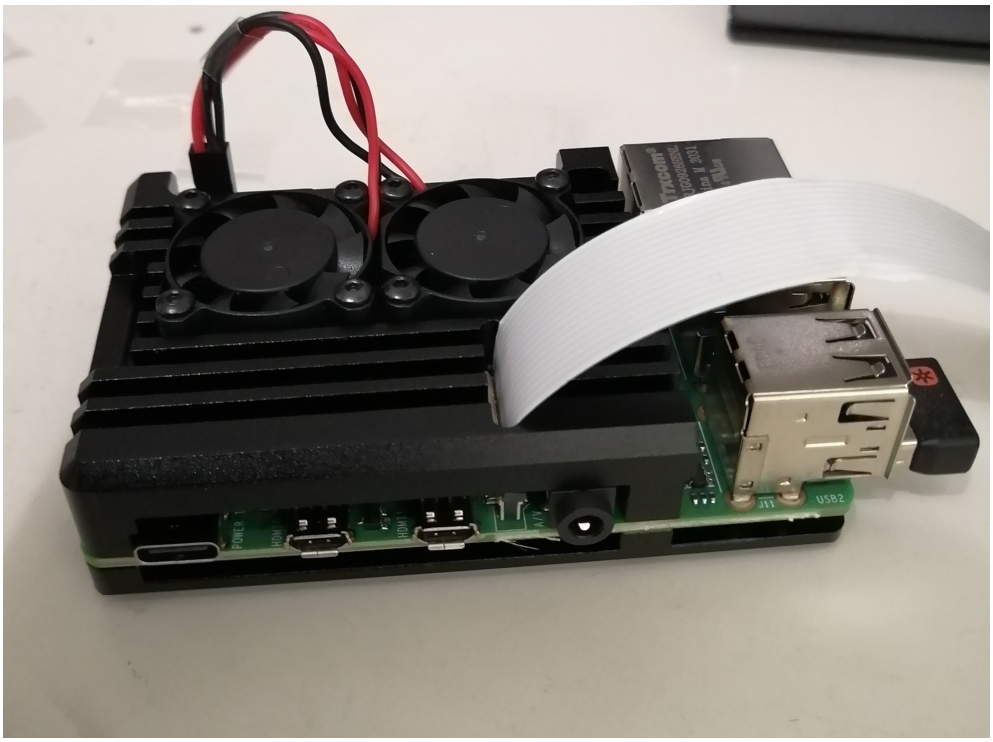


Figure 3. Cooling case

3.2. Software

The original idea was to capture an image with Python and save it at some specific location, where faceDetect.cpp could read it from. Then faceDetect.cpp would detect whether there is a face in the image and process it for recognize.cpp to recognize the

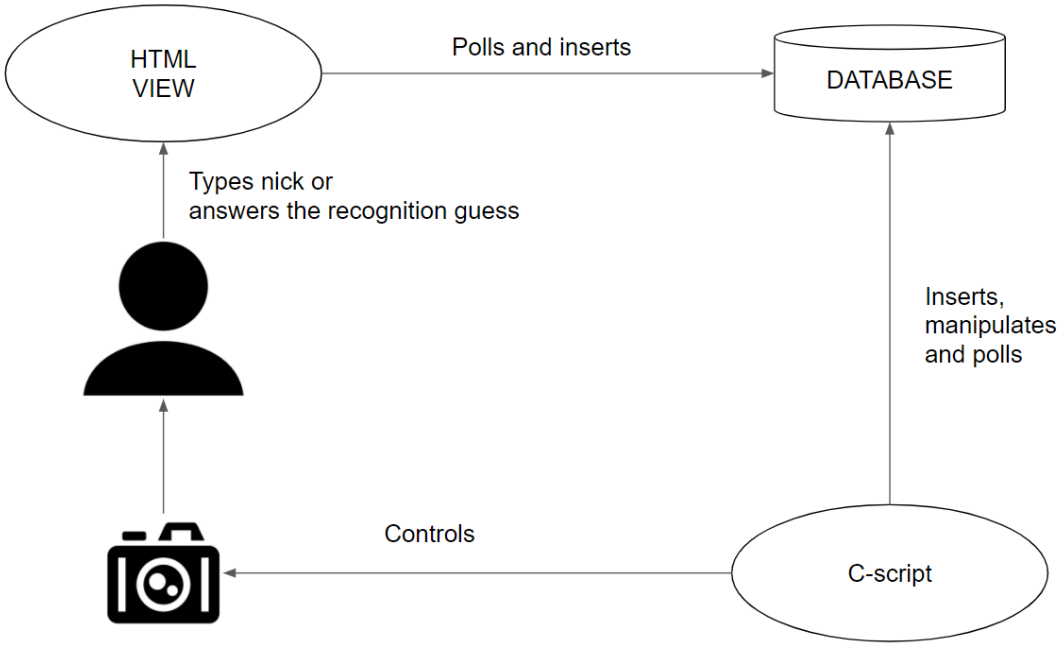


Figure 4. Simplified flowchart

person. After which, the recognize.cpp would inform main.py whether the person was recognized, and then the name would be delivered to the web interface via database.

As it turns out, OpenCV can easily capture images, which simplifies data flow significantly and reduces latency and improves performance.

While going further into our project, we dropped the idea that a python script would control everything and replaced it with a single C file that combines both faceDetect.cpp and recognize.cpp. The discard of Python was due to the problems with having C and Python working together. This change also decreased the latency a bit, as well as simplifies the development significantly. However, this in turn introduced an issue of how the C based back end and PHP based front end can communicate, as this was supposed to be done using the python layer. Utilizing a database was deemed the most suitable option.

Thus the final structure of the project is the following: first faceDetect.cpp detects if a face is seen in a picture, then it tries to recognize it and, according to what it sees or does not see, updates the database.

In Figure 5 can be seen the final structure of the recognition side. Compared to the earlier idea, multiple steps have been changed, the most important being to handle all the communication between the front- and back end with a MySQL database. In this new model, the recognition back end writes a specific number to a simple table whether it did not detect any faces from the camera, if it did or if it does not see a face. Chapter 3.8 presents the database more closely.

3.2.1. Choosing The Facial Recognition Method

The original idea was to use either Eigenface or Fisherface methods. Still, due to their technical implementation, the model could not be updated with new persons

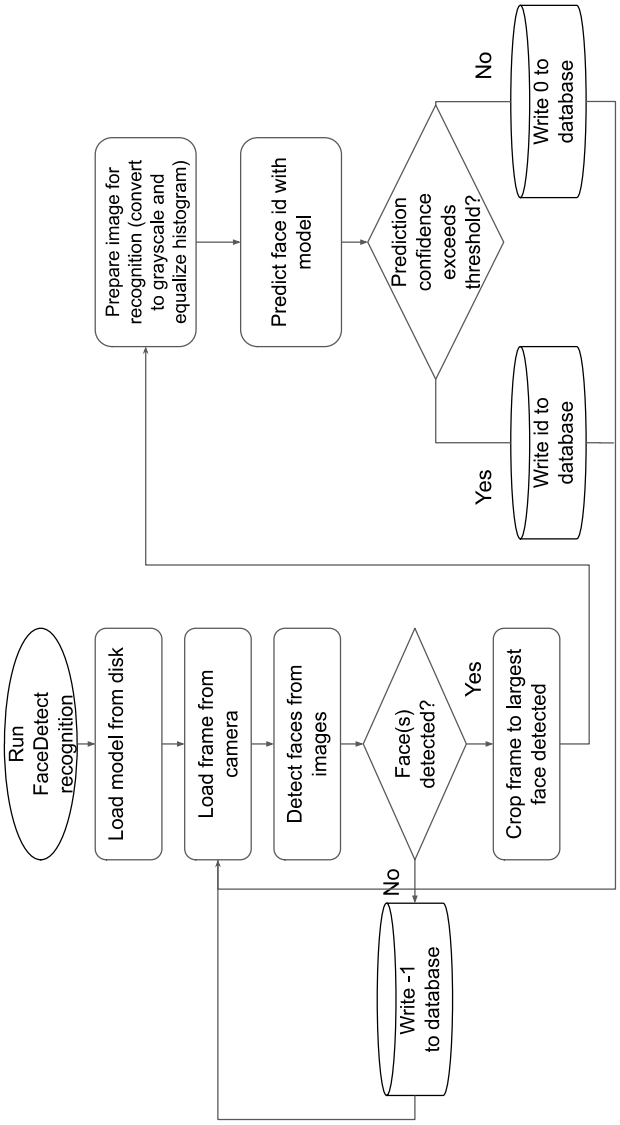


Figure 5. Flowchart of face recognition backend

without retraining the whole model. If either of these methods were to be used, all pictures would have to be saved and stored in their original format adding unnecessary complexity and performance loss from repeated retraining. On some deployments, this additional cost might be worth possible increased performance for recognition, but low-power Raspberry Pi would not be able to handle it without significant lag. Thus local binary pattern histograms was chosen as the technology since the model can be updated without complete retraining.

3.2.2. Choosing Face Detection Method

When testing different face detection cascades, lbp based ones are found to perform better than haar cascades, with fewer false positives and better detection from afar. This finding is aligned with another study [34], which measured lbp cascades to perform 140% faster and detect 4% more faces than haar cascade.

3.3. How Face Detection Works

As seen in figure 6, face detection is used to find any faces from the camera feed so that the images can be cropped to include as little excess data as possible before face recognition is attempted. Less excess data improves processing time and accuracy, as the algorithm processes only the face, not wall or other non-relevant information. In the core, face detection is implemented by using Haar feature-based cascade classifier provided by OpenCV, detectMultiScale. This classifier can be controlled by changing input parameters, such as the used classifier and scaling factor. These parameters need to be modified in the evaluation phase to optimize face detection accuracy and speed.
In this project, we utilize OpenCV's pretrained haar classifier. As this resulted in sufficient performance, there was no reason to train our own, as it would have taken a considerable amount of time for at best minor accuracy and performance gains.
Before camera feed is given to the classifier, essential image preparation is applied, including transforming to a grayscale and equalizing histogram.
After the classifier provides the coordinates of the face, only the largest face is passed to facial recognition. This face is assumed to be the closest one to the camera and thus the one that should be focused on. In future, one way to improve the software is to recognize and greet multiple people simultaneously, but current functionality is sufficient for this project.

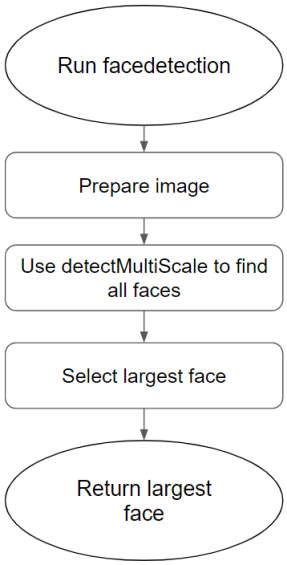


Figure 6. Flowchart of face detection

3.4. How Images Are Prepared for Recognition

Multiple methods are applied to optimise the captured images to improve face detection and recognition speed, and accuracy. Here are shown images of the original image processing seen in figure 7.

Figure 7. The original captured image [35]

The first step is cropping the original image so that only the subject's face is present, as seen in figure 8. This way, most irrelevant information such as excess background and clothing is left out, which speeds up recognition as there is less information to process. Cropping is done simply by utilizing the face detection algorithm, which provides both location and dimensions of the face.

Figure 8. Cropped version of original image

Figure 9. Grayscale version of cropped image

The next image is converted from bgr colour space to grayscale, as seen in figure 9. The conversion is done since colours have little impact on recognition performance; however, they increase the amount of data that has to be processed, thus decreasing the recognition speed.

Figure 10. Histogram equalized version of grayscale image

The last step is histogram equalization, which increases the image's contrast, bringing out the otherwise subtle patterns. This can be seen in figure 10.

3.5. Teaching New Faces

Periodically the backend checks whether there is a person in the database where id=1, meaning that the person needs to be added to the model. When this is detected, the program waits for a predetermined duration before setting the teach flag to true in

order to give the subject sufficient time to position themselves. After the flag is set, the person's face is added to the model on multiple consecutive frames.

```
Mat processedImage = prepareImage(croppedFace);
if(photo_amount_counter < photo_amount)
{
    photo_amount_counter += 1;
    cout << "Teach face with id " << std::to_string(person_id)
         << ". Photo " << photo_amount_counter
         << " out of " << photo_amount << endl;
    updateModel(processedImage, person_id);
}
else
{
    saveModel();
    teach = 0;
    person_id = 0;
    photo_amount_counter == 0;
}
```

Listing 1. Teaching a persons face over multiple frames

3.6. Face Recognition

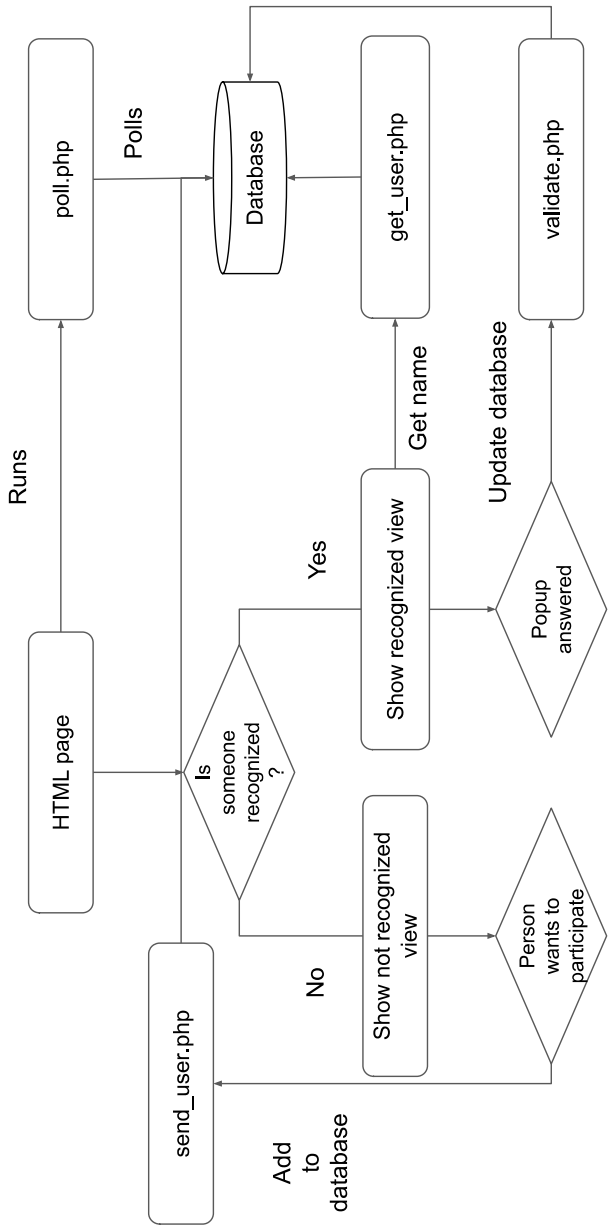
When a face is located by face detection, the face is passed for recognition after being prepared. OpenCV's predict-function is used to get the predicted id of the person, as well as the prediction confidence. As confidence tells how similar the person's face is to the most similar face in the model, its value can be used to assess whether the prediction is correct. If the confidence is lacking, the person likely does not exist in the model, in which case the frame is marked as an unknown face. Respectively, if the confidence is reasonable, the frame is labelled with the predicted id.

As the recognition confidence is given as a distance from the closest match, it is hard to define a specific confidence threshold since it changes depending on how many persons have been added to the model and how much their faces resemble each other. If the threshold is too small, the program could discard photos as not recognized if there is only a small change in appearance. This might be due to wearing glasses or changing hairstyle. On the other hand, if the threshold is set too high, the program would mislabel persons more often. This might occur with siblings, for example. In our application, we use an arbitrary value that has been deemed good enough. After testing the system in the real world, it would be possible to refine the threshold.

To improve the stability of recognition and avoid prediction changing rapidly from frame to frame if some frames are bad for one reason or another, median recognition over multiple past frames is used for reporting the predicted id. Consequently, this delays the recognition slightly, as the program has to process a set amount of frames before reporting the predicted id. This slight delay is deemed insignificant in our application.

Finally, the predicted id is pushed to the database to be retrieved by the front end.

3.7. User Interface



The front end was done with HTML, JavaScript and CSS. The web page views are very plain, and there is only a little work done appearance-wise because it is not our project's main goal. The user can see two pages depending on if the user is recognized or not. Actually, the page does not change, but the JavaScript code shows or hides different blocks on the page. See figures 11 and 12.

If the camera sees a face and if the software recognizes the face, the website will ask if the guess was correct. See figure 12. The person could press yes or no. When pressed "yes", the database will update, and the page would show the default view. See figure 11. When pressed "no", the database would also be updated, and the view would change back to default. The JavaScript code for polling can be seen in listing 2. The poll.php script that the mentioned JavaScript uses can be seen in listing 3. Other PHP

scripts are similar to this. At the end of the JavaScript script, there is a time interval for the poll, which is 1 second. This lengthens the delays for recognition, but if the time between script executes gets too short, the web page could begin to crash. All of the scripts are available in GitHub and a readme file with further installation instructions. See chapter 7 for a link.

```
<script>
var refreshId = setInterval(function(){
    $.getJSON("poll.php", function(result){
        console.log(result);
        if (result == 0) {
        document.getElementById("recognized").style.display = "none";
        document.getElementById("not_recognized").style.display = "block";

        }else if (result !=  2){
        console.log(result);
        $.ajax({
                type: 'post',
                url:"get_user.php",
                data: {
                   result
                },
                success: function(nick) {
                console.log(nick);
                document.getElementById("guess").textContent = "Are you" + nick + "?";
                document.getElementById("hidden_data").value = result;

                }
          });
            document.getElementById("recognized").style.display = "block";
            document.getElementById("not_recognized").style.display = "none";
        }
    });
}, 1000);
</script>
```

Listing 2. Javascript code for polling database

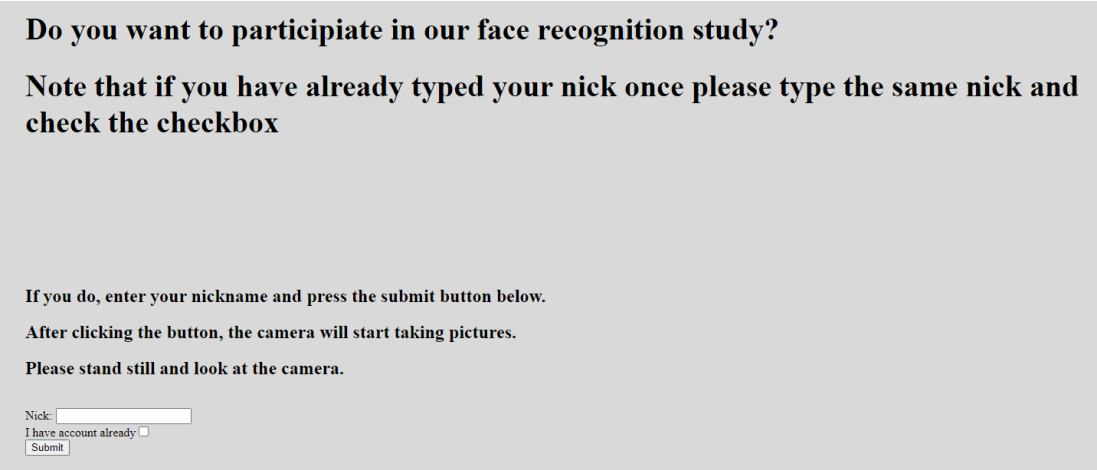


Figure 11. Image of default view

```php
<?php
$server = "localhost";
$user = "root";
$pass = "password";
$db = "recog";

$conn = new mysqli($server, $user, $pass, $db);


if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT * FROM status WHERE id=2";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
    while($row = mysqli_fetch_assoc($result))
        $user_id = $row["id"];
        $status = $row["status"];
    utf8_encode($user_id);
    if ($user_id == 2) {
        $sql = "SELECT person_id FROM test WHERE id=2";
        $result = $conn->query($sql);
        if ($result->num_rows > 0) {
        while($row = mysqli_fetch_assoc($result))
               $person_id = $row["person_id"];
           utf8_encode($person_id);
           echo json_encode($person_id);
    }
}else{
    echo json_encode(0);
}
}

$conn->close();


 ?>
```

Listing 3. poll.php

3.8. Database

C-script and JavaScript manipulate the MySQL database. C-script changes information in the status table according to what the camera sees. See Table 2. The same script also polls the database if it needs to teach the software a new face. The HTML view polls the status table with JavaScript to change the HTML view depending on if the C-script recognizes a person that the camera sees.

3.8.1. Status Table

The status table is the table that HTML view polls via PHP and is manipulated by the C-script. This table tells if we have recognized someone and who it is.

id	person_id	confidence	status
int(10)	int(10)	int(10)	int(10)

Table 2. Status table's database structure

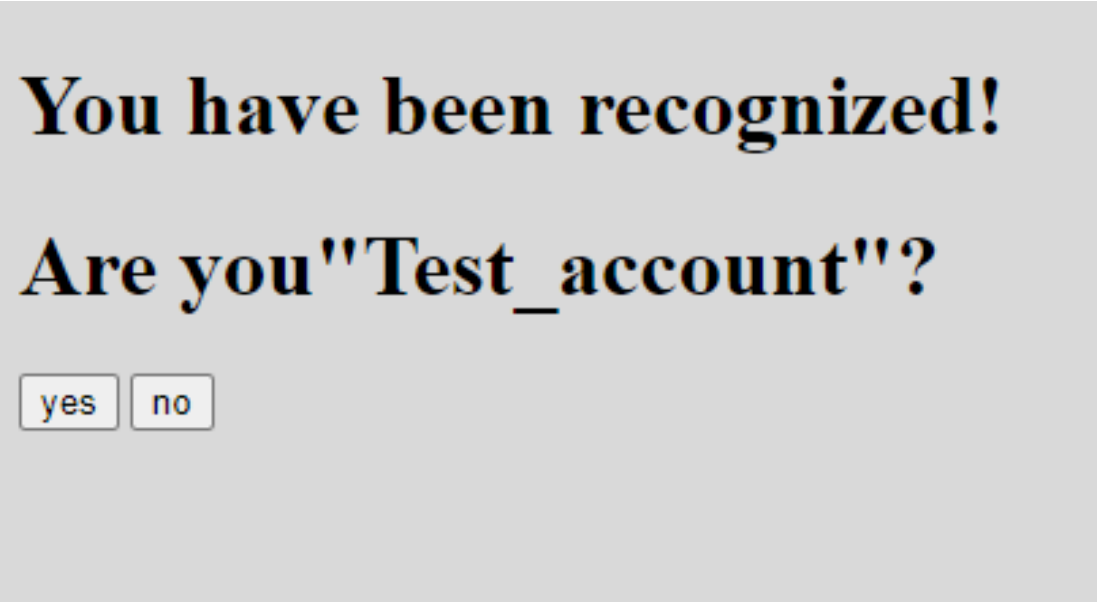


Figure 12. View if someone is recognized

The first column, "id", defines what the HTML page shows or if the software needs to train a new face. If "id" is 0, it means that the camera does not see a person. If it is 1, this tells our C-script to take pictures and train a new person's face. 2 as id means that the camera sees a person. The second column, "person_id", tells us the recognized person's id. This is the same as the person's id in the users table. This way, we can connect the person_id with a nickname. "Confidence" is a value that tells how sure the software is about its recognition. The "Status" is connected with the "id" column. When the id is 2, we can have 0 or 1 as a status value. 0 means that the software does not recognize the person the camera sees, and 1 means that the software has recognized the person.

3.8.2. Users Table

Users table consists of saved users' nicknames and data of how well they were recognized.

id	nick	guesses	answers	corrects
int(11)	varchar(20)	int(50)	int(50)	int(50)

Table 3. Users table's database structure

The first column, "id", is auto-incremented when a person creates a nickname via the HTML view. The second column, "nick", is the nickname a participant chooses to have. The "guesses" contains the number that the software has guessed this person. The "answers" column is the amount of how many times the question "Are you this person" has been answered on the website. This number could be used to see how interactive people are with this system by comparing this number to guesses' column's number. The "corrects" column is the amount of how many times the software has

made a correct recognition for this person.

3.9. System Setup

All of the scripts and the web page server run locally on the Raspberry Pi. As an HTTP server, we are using Apache version 2.4.38 and as a database MySQL. There was not much comparison to other HTTP servers or databases. These platforms that we use were chosen because they were suitable and lightweight enough for Raspberry Pi.

PHP version 7.3.14 was installed because of the communication between the database and the HTML views. At the beginning of our project, there was a discussion if web frameworks like Flask or Django would be installed. We chose not to install any web framework because our project is small enough to just write everything in plain HTML and PHP.

Regarding face recognition technology, multiple technologies were investigated, notable ones being dlib, OpenFace [17] and OpenCV [30]. C++ based OpenCV library was chosen for its seemingly easiest usage and performance, so its latest version 4.2.0 was installed on the system.

4. EVALUATION

The main goal of our evaluation process is to determine how reliable our face recognition software is. We were also going to study how much people would interact with our project, but it was not done since the University of Oulu closed its doors due to ongoing COVID-19. Our first evaluation plan was to install our setup on a busy corridor at the university, but this plan was changed due to the same reason. The newer plans are discussed in the next section.

4.1. Plan

As mentioned before, we want to know how well our face recognition works. Two different tests are planned and carried out. The first one is to learn how good the face recognition software is using an existing dataset of facial images, FERET database [9] by National Institute of Standards and Technology. Furthermore, the other evaluation part is a smaller test to learn whether everything else works as expected in real life. For example, can we teach the software with images given from Raspberry Pi's camera?

The other part is done with people as test subjects. We evaluate how the confidence level affects the recognition accuracy and whether the accuracy changes if a person is wearing glasses.

4.2. Theoretical Evaluation

The theoretical evaluation is done using the FERET dataset to be able to assess program performance and accuracy with a large number of images. Since it is necessary to run large number of tests in order to evaluate comprehensively how different variables affect outcome, a more powerful computer with i7-7600U processor is used to speed up the process. This results in framerate being approximately 50% less than achieved in tests. The training is done using pictures of 994 persons recommended by FERET. Testing uses two different datasets, one containing pictures taken immediately after training pictures, fb and one where pictures were taken later, dup1. One test is also executed using images from multiple datasets combined, fb, dup1 and dup2. These test results are then used to tweak algorithm variables to optimise recognition speed and accuracy.

To evaluate the effects, different variables have on performance and accuracy, an evaluation formula is created to find a good balance between accuracy and performance. The formula multiplies feature scaled values of accuracy and fps to evaluate all test results between 0 and 1 inclusive.
The feature scaling formula used is

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{1}$$

where min and max get smallest and largest values respectively across all tests run. The evaluation formula created is

$$y = fps' * accuracy' \tag{2}$$

This approach has several caveats since it differs significantly from the intended application. Consequently, the results are mainly indicative of real-world performance. The most considerable difference is the lack of multiple pictures; the test consists of only one training and one testing photo of each person. Our program takes multiple training photos and uses multiple photos to improve recognition accuracy. Therefore, even though it would seem best to sacrifice recognition speed for accuracy in theoretical evaluation, this would result in a smaller amount of photos taken in a real-life test and thus might lead to worse accuracy overall.

Another caveat is the lack of retraining. In the real world, our program would be retrained whenever a person is being recognised incorrectly, and thus recognition accuracy improving over time. In this theoretical test, no retraining happens.

4.2.1. Process

To run tests unsupervised, the testing process is automated by creating two scripts that create CSV files with paths of the images to use on each run, as well as run the test repeatedly, tweaking variables between each run, and adding results to the log CSV file. Neighbours and radius variables were deemed to have the most significant correlation with performance and accuracy, so their effect was investigated more thoroughly. They are later analysed further utilising R-language.

FERET's specification is followed for the first round of tests to get a baseline: one training image of each person and treat each testing image individually. This is done to assess what effects our tweaks of using multiple images have on the algorithm, either positive or negative. The tests are run with radius values ranging from 1 to 30, and neighbours ranging from 1 to 10. Not all values are tested between these ranges to reduce processing time. The tests are performed using two testing datasets, one taken immediately after training photo (fb) and one where images were taken at least a couple of days later (dup1).

For the second round of tests, the algorithm is tweaked to match how the program works in real life, meaning that all sequential images of one person are used to calculate the median recognition. Even though this is not how the tests are intended to be run by FERET, this gives us better insight into how our program would fare in a real-world deployment. The tests are run using a single dataset similar to the first tests and using all three datasets combined and sorted so that all pictures of the same person are back-to-back. The test results can be found from table 5, where the first column *Multiple images* tells whether our modifications utilising multiple images is being used.

4.2.2. Results

Overall, the theoretical testing provides useful insight into the programs' accuracy and how different variables affect recognition performance and accuracy. Of the two variables tested, neighbours has a much greater impact on performance and accuracy than the radius. Pearson correlation coefficient between variables and results can

	Radius	Neighbours
Accuracy	0.0633	0.8452
FPS	0.0159	-0.9402

Table 4. Pearson correlation coefficient between radius, neighbours, accuracy and FPS

Multiple images	Dataset	Accuracy	FPS	Radius	Neighbours
No	fb	96.27%	2.01	20	10
No	fb	93.84%	13.74	20	5
No	dup1	50.20%	1.74	14	10
No	dup1	43.27%	14.15	14	5
Yes	dup1	60.00%	1.73	14	10
Yes	dup1	52.80%	13.59	20	5
Yes	All	90.42%	1.98	20	10
Yes	All	86.49%	13.34	20	5

Table 5. Relevant results from theoretical tests

be seen on table 4. These two variables need to be balanced to find optimal values depending on deployment location, the hardware used and desired recognition speed. How neighbours and radius affect evaluation formula, can be seen on figures 13 and 14. In our scenario, the evaluation formula is utilised to choose neighbours value of 4. Radius had only a small effect in our tests, but 10 had a slight advantage over other values tested.

As expected, dup1 dataset is much more difficult for the algorithm, but it is also more realistic, given that subjects had different clothing, hairstyle, expression etc, similar to real life. For the first round of tests, treating each picture individually, the best recognition accuracy reached is 96.27% when neighbours = 10 with fb-dataset and 50.20% using dup1-dataset, as seen on rows two and four on table 5. The recognition framerates were 2.01 and 1.74, respectively. Framerate this low is an issue, so a better alternative would be to use 5 neighbours to increase framerate 600-800% and sacrifice accuracy 4-14%, as seen on rows three and five. Even though the accuracy is seemingly lower, the program has time to analyse more pictures of the subject, thus increasing the amount of data that can be gathered, improving accuracy.

For the second round of tests, using the easiest dataset, fb, no relevant change in accuracy nor performance is detected, so data is not included. This is probably due to the easy nature of the dataset. With more difficult dup1-dataset, accuracy improvement of 8-10 percentage points is detected to a maximum of 60.00% when neighbours = 10 in comparison to the first round of tests, without significant effect on performance. Again, using neighbours = 5 improves fps significantly, 780% to 13.59, but accuracy decreases only 12% to 52.80%. These can be seen on rows six and seven on table 5. This suggests that using multiple images for recognition improves accuracy. The only obvious downside is that there is a longer delay, as more frames need to be analysed before the result is shown to the user. When using all three datasets, the accuracy of 90.42% is reached when radius = 20 and neighbours = 10. Similar to other datasets, a neighbours value of 5 would probably be a better choice since accuracy decreases only 4.35%, while framerate increases 574%. These can be seen on rows eight and nine on table 6.

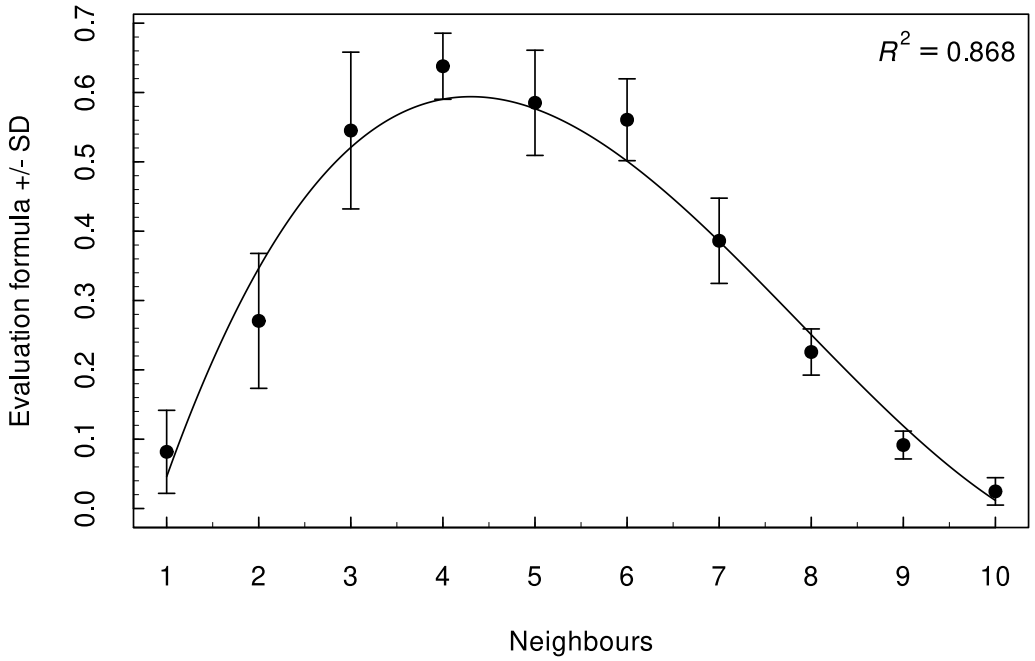


Figure 13. Neighbours vs evaluation formula

Scaling of the images before recognition is also tested, but a couple of test runs showed that scaling decreased accuracy significantly, with only a minor improvement to fps. Thus scaling is deemed not to be worth using.

4.3. Hands-On Test

4.3.1. Process

The hands-on test was carried out in a room with good lighting conditions. Four different tests were carried out with three participants. The first test was with a confidence level of 500. The second test used the same confidence level, but the participants were wearing glasses. The third test was done with a confidence level of 1000, and the fourth test with a confidence level of 10000. The confidence limit is the faceDetect.cpp's variable "confidence_limit", which defines how good the software's guess has to be to accept it.

In the beginning, every participant's face was taught to the software and added to the database. The participants did not wear glasses during the teaching. The same picture taken in the beginning was used for every test. After taking the teaching pictures, the participants went in front of the camera in random order. The distance between participants and the camera varied from one meter to two meters. The participants tried

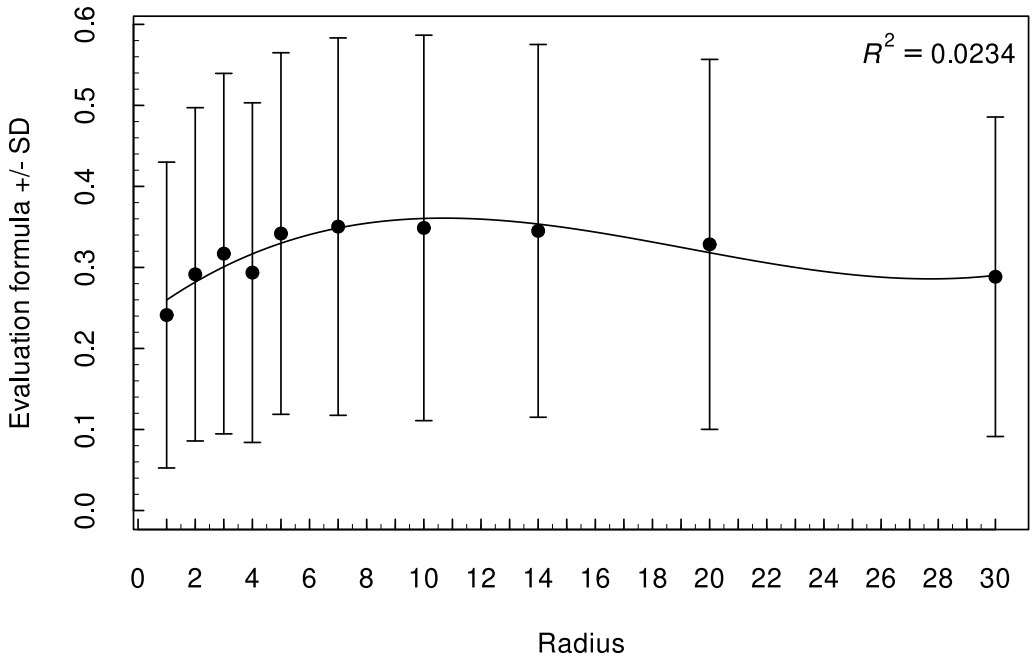


Figure 14. Radius vs evaluation formula

to stay still, but some movement during tests happened. Twenty software's outputs were taken from every participant, which totals sixty outputs for each test.

During each test, one person wrote down the wrong and right guesses. This was done to double-check the results with the database. The participant in front of the camera used the keyboard to tell the software if the guess was right or wrong. The test setup can be seen in the figure 15.

4.3.2. Results

The Table 6 shows the results from the hands-on tests. The first column is the number of the test. The second column shows how many correct guesses the software made. The third column shows how many wrong guesses it made. In these tests, the wrong guess means that the software guessed someone else. The column "confidence" stands for the variable "confidence_limit" in faceDetect.cpp that was manipulated between tests. Note that test number two was carried out with participants wearing glasses. Other tests than the number two were performed without glasses. The glasses used in this test can be seen in figure 16.

From table 6, can be seen the effect of confidence on the accuracy. By setting the confidence limit higher, the recognition delay and accuracy decreased both. With all of the different confidence limits, the recognition time was from around 1 second to a couple of seconds. The difference between latencies came from the web page code. With a lower confidence level of 10000, the delay is slightly lesser than when using a

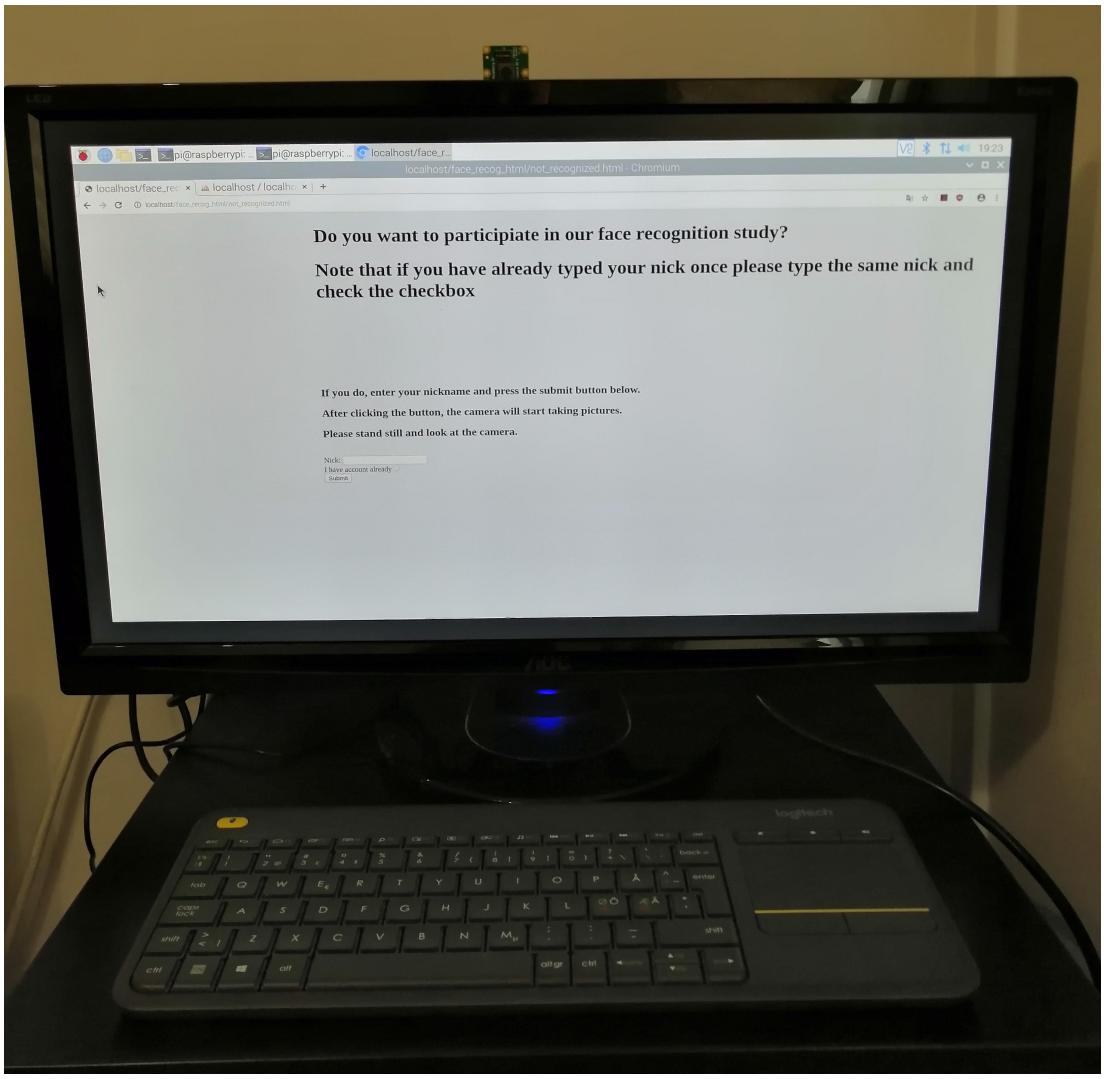


Figure 15. Test setup

confidence level of 500. The front-end, in this case, the Javascript function, gives an additional delay because it executes the polling function every second. In test number one, the accuracy was good, but in test number 2, when participants wore glasses, the accuracy dropped remotely. While carrying out the tests, it was noticed that if participants were standing still and looking directly into the camera, the recognition was faster. It could also be argued that the software made more mistakes about the person if the participant moved around.

From these results, the confidence limit has to be chosen carefully depending on the use case. By dropping the confidence level under 1000, the accuracy will rise, but the recognition time will be slightly longer. Raising the confidence will decrease the accuracy, but the delay will be slightly shorter.

Since a person needs to stand still in front of the camera for a second and answer the software, it could be argued if this is a fitting solution when the recognition is done to people who are just casually walking past the camera. In situations where people have the time to stop for a moment, this implementation could be a good solution for face recognition.

Figure 16. Test glasses

Test number	Correct	Wrong	Confidence
1	59	1	500
2	54	6	500
3	44	16	1000
4	39	21	10000

Table 6. Results

5. SUMMARY

To summarise this paper, our implementation seems to work well with a decent accuracy even though the amount of tests in the real world needs to be noted. Raspberry Pi 4B is a working platform for a face recognition system, but its parts need to be handled carefully, and it has to be cooled to prevent possible overheating.

Multiple improvements could be made. A lot of them have to do to interaction. We could install the system into a place where people could use our system. In its current state, our system is not very compelling to people to use it. To make it more compelling, the views could be more polished, and the colours could be distinctive. Also, a sound system could be connected to the Raspberry that it would, for example, say "Hello" if it recognizes someone. After installing the system into a public space, further tests could be done as well.

Another idea we had was that a person could create an avatar for him/herself which would also pop up every time the camera recognizes her/him. This feature could be done, for example, with JavaScript.

If this process were to be continued further, it would be good to test different scaling values and different classifiers. Unfortunately, this would lead to a significant increase in data output and the number of tests that would have to be run. We estimate that with our current setup, it would take approximately seven years to run enough tests to reach meaningful accuracy. This would lead to the need to use a coarse-to-fine approach and run the first few tests with sparse variable values to narrow down the variable range to be tested further or reduce testing duration in some other way.

This would lead to the need to be able to analyze multi-dimensional test data, so procedures like Multivariate analysis of variance would have to be used to find out what kind of effect different variables have on both accuracy and framerate, as well as how they affect each other. Analyzing results is complicated further because some classifiers might work better with specific variable values than others.

If scaling is implemented, it might be worthwhile to adjust the scaling amount based on the original size of the image. For example, if the original image is 700px wide, it would be scaled more than a 100px wide one. This is because scaling is done only to reduce the processing time of large images, so scaling small images is unnecessary.

Using nested cascades could provide exciting results. Nested cascades search items inside another cascade, for example, trying to find eyes in the area where the face should exist. Frames, where no objects can be found, could be discarded, as image quality might be lacking, or the person's face is not entirely visible. This would improve both teaching and recognition as bad images do not "confuse" the model.

Teaching new faces could also be improved by comparing sequential images taken for their similarity. If some frames differ significantly from others, it would suggest that maybe the image is of another person, the persons face is occluded, or the image quality is lacking. Subsequently, defective frames could be discarded and retaken, thus improving model accuracy.

As good values for the confidence limit are affected by how many faces have been added to the model; it would be good to adjust the limit dynamically. The easiest solution would be to create some formula that decreases the confidence limit as more faces are added, but finding good formula could be difficult. A bit more elegant solution is to decrease it when someone is identified incorrectly and respectively

increase the limit when someone is not recognized despite being taught to the model. If the face data was saved, it could be possible to backtest how different confidence limit values would have fared in previous recognition events.

Unlike in most facial recognition papers, we benefit from using a video feed instead of individual images. This has the benefit of having more data to recognize a person. As we did not account for this initially, this resulted in issues if a person is not recognized reliably on every frame. This resulted in unpredictable results, as it depended entirely on when the recognition data was read from the database. This issue was resolved and rigidity improved by using median recognition past multiple frames. For example, if over a period of ten frames, the software recognizes person A on six frames and does not recognize anyone on four frames, it can be assumed that indeed person A is in front of the camera. The downside of this approach is an increased delay between first introducing a person and recognition, but even if ten frames are used, the delay is negligible 166 ms.

From the hands-on tests, one can see that accurate results are received with a lower confidence limit. The downside to this is that the delay grows slightly when Raspberry Pi is used as a platform. If the confidence limit is set higher, the accuracy decreases. Future work could be on solving how to decrease the delay without losing accuracy. To achieve this, one would need to look at the face recognition variables, chosen cascade, variable confidence limit in faceDetect.cpp, the median recognition and how it affects the delay and accuracy, and front-ends JavaScript polling delays.

Recognition variables could also be adjusted dynamically, as users input whether they have been recognized correctly. This would be most useful in defining the appropriate confidence threshold.

6. REFERENCES

[1] Espoossa testataan kasvojentunnistusta läsnäolon seurannassa (2019). URL: https://yle.fi/uutiset/3-10630994. Accessed 07.10.2020.

[2] FA6 MED – FACE RECOGNITON FOR HOSPITALS. URL: https://www.face-six.com/patient-identification/. Accessed 08.10.2020.

[3] Google Nest Hello . URL: https://store.google.com/fi/product/nest_hello_doorbell. Accessed 07.10.2020.

[4] Turk M. & Pentland A. (1991) Eigenfaces for recognition. Journal of cognitive neuroscience 3, pp. 71–86. Accessed 1.11.2019.

[5] Belhumeur P.N., Hespanha J.P. & Kriegman D.J. (1997) Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. IEEE Transactions on Pattern Analysis and Machine Intelligence 19, pp. 711–720. Accessed 1.11.2019.

[6] Ahonen T., Hadid A. & Pietikäinen M. (2006) Face description with local binary patterns: Application to face recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 28, pp. 2037–2041. Accessed 1.11.2019.

[7] Hassner T. (2013) Viewing real-world faces in 3d. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 3607–3614. Accessed 1.11.2019.

[8] Nest Hello. URL: https://support.google.com/googlenest/answer/9243617?hl=en. Accessed 1.8.2021.

[9] Phillips P.J., Moon H., Rizvi S.A. & Rauss P.J. (2000) The feret evaluation methodology for face-recognition algorithms. IEEE Transactions on Pattern Analysis and Machine Intelligence 22, pp. 1090–1104. Accessed 1.11.2019.

[10] Brunelli R. & Poggio T. (1992) Face recognition through geometrical features, vol. 588 LNCS. Springer Verlag, 792-800 p. Accessed 1.11.2019.

[11] Adini Y., Moses Y. & Ullman S. (1997) Face recognition: The problem of compensating for changes in illumination direction. IEEE Transactions on Pattern Analysis and Machine Intelligence 19, pp. 721–732. Accessed 1.11.2019.

[12] Taigman Y., Yang M., Ranzato M. & Wolf L. (2014) Deepface: Closing the gap to human-level performance in face verification. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 1701–1708. Accessed 1.11.2019.

[13] Hassner T., Harel S., Paz E. & Enbar R. (2015) Effective face frontalization in unconstrained images. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 07-12-June-2015, vol. 07-12-June-2015, pp. 4295–4304. Accessed 1.11.2019.

[14] Zhang K., Zhang Z., Li Z. & Qiao Y. (2016) Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Processing Letters 23, pp. 1499–1503. Accessed 1.11.2019.

[15] Zhu X., Lei Z., Liu X., Shi H. & Li S.Z. (2016) Face alignment across large poses: A 3d solution. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2016-December, vol. 2016-December, pp. 146–155. Accessed 1.11.2019.

[16] Schroff F., Kalenichenko D. & Philbin J. (2015) Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 07-12-June-2015, vol. 07-12-June-2015, pp. 815–823. Accessed 7.10.2020.

[17] Baltrusaitis T., Robinson P. & Morency L.P. (2016) Openface: An open source facial behavior analysis toolkit. In: 2016 IEEE Winter Conference on Applications of Computer Vision, WACV 2016. Accessed 07.10.2020.

[18] Zhao W., Chellappa R., Phillips P.J. & Rosenfeld A. (2003) Face recognition: A literature survey. ACM Computing Surveys 35, pp. 399–458. Accessed 1.11.2019.

[19] Vijayalakshmi M., Thulluri K.V. & Kanchana C.S. (2019) Face recognition door unlock system. International Journal of Innovative Technology and Exploring Engineering 8, pp. 1133–1139. Accessed 9.11.2019.

[20] Jeon B., Jeong B., Jee S., Huang Y., Kim Y., Park G.H., Kim J., Wufuer M., Jin X., Kim S.W. & Choi T.H. (2019) A facial recognition mobile app for patient safety and biometric identification: Design, development, and validation. Journal of Medical Internet Research 21. Accessed 9.11.2019.

[21] Neto L.B., Grijalva F., Maike V.R.M.L., Martini L.C., Florencio D., Baranauskas M.C.C., Rocha A. & Goldenstein S. (2017) A kinect-based wearable face recognition system to aid visually impaired users. IEEE Transactions on Human-Machine Systems 47, pp. 52–64. Accessed 9.11.2019.

[22] Sukmandhani A.A. & Sutedja I. (2019) Face recognition method for online exams. In: 2019 International Conference on Information Management and Technology (ICIMTech), vol. 1, vol. 1, pp. 175–179. Accessed 9.11.2019.

[23] Liao G., Chen W. & Wu Y. (2016) Facial features for gender recognition. In: 2016 35th Chinese Control Conference (CCC), pp. 4161–4165. Accessed 9.11.2019.

[24] Kumar S., Singh S. & Kumar J. (2017) A comparative study on face spoofing attacks. In: 2017 International Conference on Computing, Communication and Automation (ICCCA), pp. 1104–1108.

[25] Erdogmus N. & Marcel S. (2013) Spoofing in 2d face recognition with 3d masks and anti-spoofing with kinect. In: 2013 IEEE Sixth International Conference on Biometrics: Theory, Applications and Systems (BTAS), pp. 1–6.

[26] jquery (2020). URL: https://jquery.com. Accessed 16.8.2020.

[27] htmldom (2020). URL: https://www.w3schools.com/js/js_htmldom.asp. Accessed 17.8.2020.

[28] Css w3schools (2020). URL: https://www.w3schools.com/css/css_intro.asp. Accessed 17.8.2020.

[29] jquery w3schools (2020). URL: https://www.w3schools.com/jquery/jquery_intro.asp. Accessed 16.8.2020.

[30] Bradski G. (2000) The OpenCV Library. Dr. Dobb's Journal of Software Tools .

[31] Mysql (2020). URL: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html. Accessed 17.8.2020.

[32] Mysql intro (2020). URL: https://dev.mysql.com/doc/refman/8.0/en/introduction.html. Accessed 16.8.2020.

[33] Apache (2020). URL: https://httpd.apache.org/ABOUT_APACHE.html. Accessed 17.8.2020.

[34] Kadir K., Kamaruddin M.K., Nasir H., Safie S.I. & Bakti Z.A.K. (2014) A comparative study between lbp and haar-like features for face detection using opencv. In: - 2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T), pp. 335–339. ID: 1.

[35] President Barack Obama. URL: https://fi.wikipedia.org/wiki/Barack_Obama#/media/Tiedosto:President_Barack_Obama.jpg. Accessed 1.5.2021.

7. APPENDICES

Appendix 1 The source code can be found from here: `https://github.com/aivalja/space-greeter`